Benson Chu

11/11/18

CS 165A

MP 1 Report

Architecture

- My code architecture includes an array of structs, and in each of these structs, there is some information regarding the specific class, including P(C), the total number of documents within this class, and an unordered map that stores all the words in each document within this class, with the value of each word being its frequency. The index of the array denotes the document number. This array of structs is an array of the different documents' labels (1 – 15).

Preprocessing

- When reading each document and storing the values in the unordered map for the corresponding document, I made sure not to include various stop words, which were stored in a set.
- Regarding the representation of a document, I defined a document as being the set of all documents within the same class. Each non-stop word was stored as a key in the unordered map. This means that each key in the unordered map for the class is a feature.

Model Building

- The Naïve Bayes Classifier was trained by parsing the unknown document and storing its non-stop words in an unordered map with (word, frequency) pair. I would iterate through each class, and each of these classes would iterate through the entire unordered map of the unknown document, where the class with the highest value (after calculating $P(x_1 \mid c) * P(x_2 \mid c) * \ldots * P(c)$) will be used to classify the unknown document.

Results

- The results on the datasets are:
    - Training: 55% accuracy with 2 seconds training time
    - Testing: 47% accuracy with 2 seconds labeling time
- The 10 most important features for each class are the 10 values in the key-value pairs of the unordered map in each of the classes. This is because the highest frequencies in each of these unordered maps has a large impact on the probabilities calculated using the Naïve Bayes Classifier. Ex) Assuming the word is present in class c, the probability of a word in an unknown doc for a specific class is $P(x_{word} \mid c) = (freq_{word}$ in c) / (total number of words in c + ($freq_{word}$ in c – $freq_{word}$ in unknown document)).

Challenges

- There were some challenges when attempting this machine problem, of which consist of:
    - Cleaning the training documents
        - I contemplated on whether or not I should use the stemming of words, but since I was unsure of whether or not I was able to use a library from another person's GitHub, I decided to instead cleaning the training and testing documents by removing English stop words.
    - Storing the documents by class
        - Figuring out how to store the documents took some time. I was considering various ways store information of each class such as using which structure (unordered map, set, etc.) to store the values of each document. Also, I was deciding on whether or not I should have a dictionary for each document under the specific class, but in the end, I decided to combine all the documents under the specific class by storing the words in each document in a large unordered map. Each class was assigned based on the index of the array of structs (1 – 15), and each class had its own dictionary (unordered map).

Weaknesses

- A weakness in my method include not using stemming (grouping of similar words), since this will affect the labeling of many documents. Ex) Unknown document should be assigned to Class 1 because Class 1 has many uses of the word "excite, exciting, excited", but the classifier will choose Class 2 because Class 2 has the most uses of the word "excite", which is the only variation of "excite" in the unknown document.
    - A way to overcome this is to add stemming library to group the similar words together while both training and classifying the text files.
- Another weakness in my method includes not accounting for whether or not a word is in multiple documents. This is a large problem because if a conglomerate of words in each class consist of a document with a lot of repetition of the same word, then the classification wouldn't be accurate if the correct class has more of these words in various documents. A solution to this is to add another unordered map that stores the word -> number of unique occurrences in the total number of documents and account for this when calculating the probabilities.